Metric and Topological Entropy

Megan Lynn Tucker

December 11, 2019

1 Introduction

For this project, Matlab code was developed to compute both metric entropy and topological entropy of sequences of letters. The functions were used to calculate the entropy of a set of DNA.

1.1 Metric Entropy

"Entropy is a measure of information content and complexity," states Koslicki [1]. It has widespread applications, but here we focus on DNA; essentially, we wish to measure the entropy of strings of letters.

Suppose we are given an alphabet, $\mathcal{A}$ (for DNA sequences, $\mathcal{A} = \{A, C, G, T\}$). We define the word or string w as a finite combination of elements of $\mathcal{A}$. The set of all n-length words formed from $\mathcal{A}$ is denoted by $\mathcal{A}^n$ and $\mathcal{A}^* = \cup_{n\geq 0}\mathcal{A}^n$. Lastly, let $|w|$ denote the length (cardinality) of the string w [1].

Claude Shannon defined metric entropy as dependent on a measurement [2]. More specifically it depends on the probability that certain letters of $\mathcal{A}$ occur in w. For example, suppose our alphabet is $\mathcal{A} = \{A, C, G, T\}$. Let w be some word defined on the alphabet (a DNA sequence). The probability A appears in w is P_A, the probability C occurs in w is P_C, and so on. Therefore the metric entropy of w is

$$H(w) = -(P_A log(P_A) + P_C log(P_C) + P_T log(P_T) + P_G log(P_G))$$

where *log* is the base e (natural) logarithm [1]. Note that in Matlab, *log* is the natural logarithm [4].

Definition 1. *Let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ and P_{a_i} be the probability a_i occurs in $w \in \mathcal{A}^*$. The metric entropy of w is*

$$H(w) = -\sum_{i=1}^{n} P_{a_i} log(P_{a_i}).$$

For example, suppose $w = AAAACCCGGT$. Then $|w| = 10$, $P_A = 4/10$, $P_C = 3/10$, $P_G = 2/10$, and $P_T = 1/10$. Therefore

$$\begin{aligned} H(w) &= -\left(\frac{4}{10}log\left(\frac{4}{10}\right) + \frac{3}{10}log\left(\frac{3}{10}\right) + \frac{2}{10}log\left(\frac{2}{10}\right) + \frac{1}{10}log\left(\frac{1}{10}\right)\right) \\ &= -1.279854225833667467182185762149429928131354186921692886 39... \end{aligned}$$

Note that metric entropy is not necessarily bounded between 1 and 0. Furthermore, the metric entropy for $w = ACCCCCCCGT$ would be different than the example whereas the metric entropy for $w = ACCGGGTTTT$ would be the same. This is because in the second one the distribution of letters in the word is identical to the example whereas in the first the distribution is different.

1.2 Topological Entropy

Unfortunately, metric entropy does not compare entropy for words of different length. This is problematic because in larger sequences, the probability that multiple letters or sequences of letters are repeated is higher. An alternative method to describe complexity is topological entropy. Before defining topological entropy, we must define the complexity function [1].

Definition 2. *The complexity function $p_w : \mathbb{N} \to \mathbb{N}$ of a sequence w is the number of different n-length subwords (with overlaps) appearing in w, i.e.*

$$p_w(n) = |\{u : |u| = n \text{ and } u \text{ appears as a subword of } w\}|.$$

Note that we define subwords to be groups of letters found in the original alignment of the word. Let us break down what $p_w(n)$ is. We want to find all u such that its length is equal the n. This the cardinality of this list which is a single number.

Definition 3. *Let w be a finite sequence of length $|w|$ on the alphabet $\mathcal{A}$ and n be the unique integer such that*

$$4^n + n - 1 \leq |w| < 4^{n+1} + (n+1) - 1$$

Then for $w_1^{4^n+n-1}$, the first $4^n + n - 1$ letters of w, the topological entropy of w is defined as

$$H_{top}(w) = \frac{log_4(p_{w_1^{4^n+n-1}}(n))}{n}$$

where the base 4 logarithm is used.

Note that $w_1^{4^n+n-1}$ indicates we are using the value of n satisfying the above inequality. For example, let $w = AAAAACCCCCTTTTTTTTTTG$. Clearly $|w| = 20$. We can easily solve the inequality by examination and find that $17 = 42+2-1 \leq |w| < 43+3-1 = 66$. Therefore $n = 2$, meaning we must find all subwords of w of length 2: $\{AA, AC, CC, CT, TT, TG, GG\}$. The cardinality of this set is 7, so $p_w(2) = 7$. Recall we defined subwords as words found in the original alignment of the word. So, although CA is a possible combination of letters in w, it is not a subword.

$$H_{top}(w) = \frac{log_4(7)}{2} = 0.701838730514401026860492329307957702160256656491535195919...$$

Note that the topological entropy is the same for the sequence $AAAAACCCCCTTTTTTGGGGG$ despite the number of T's and G's differing. Similarly, the topological entropy of $AAAAACCCCCTTTTTTGG$ is the same despite $|w| = 17$. This is because the inequality yields the same value for n.

As we have defined it, $H_{top}(w)$ is always bounded by 0 and 1. Sequences with lower complexity have topological entropy closer to 0; the converse is true for more complex sequences. Again, we can compare the topological entropy of sequences of different lengths whereas with metric entropy we cannot [1].

2 Code

2.1 MetricEntropy

Define a function MetricEntropy(w, alphabetSize) that computes the metric entropy of w if it is assumed that the alphabet has size alphabetSize (that is $|A|$ = alphabetSize).

We begin by assuming we are dealing with any alphabet, not just the A, C, T, G alphabet for DNA sequences. Therefore, we must first parse through the word to find which letters compose the alphabet. This process is completed by the first for loop in the code below. The functions, `extractBetween` and `extractBefore` were used [4]; `extractBetween` "extracts the substring that begins with the first character of `str` and ends before `endStr`" and `extractBefore` "extracts the substring from `str` that occurs between the substrings `startStr` and `endStr`". The prebuilt function `contains` was also used to check if a given letter was already counted; `contains` "returns 1 (true) if `str` contains the specified pattern, and returns 0 (false) otherwise"[4].

The second for loop counts how many times each letter appears in the word. The number of occurrences of each letter are input into a blank vector for later use. This function makes use of the `count` function which "returns the number of occurrences of pattern in `str`"[4].

The last for loop calculates the probability of each letter in the alphabet by dividing the number of occurrences of each letter by the total number of letters. It then loops through each letter to calculate the metric entropy.

```
function M = MetricEntropy(w, alphabetSize)
%note that w must be a character array

    letterList = {};
    %blank character array; will hold all values of w with no repeats
    counter = 1; %itterates through letterList
    for i = 1:alphabetSize
        letter = extractBetween(w,i,i); %i'th value of w
        prevLetters = extractBefore(w, i);
        %creates a character array of all values of w before i
        if contains(prevLetters,letter) == 0
            %checks if prevLetters contains letter
            letterList{counter} = letter;
            %if not, input that character into letterList
            counter = counter + 1; %increase counter
        end
    end

    letterTotal = [];
    %blank vector; holds number of occurances of each letter in letterList
    for i = 1:(counter - 1)
        %counter holds one more value than length of letterLis; subtract 1
        letterTotal(i) = count(w, letterList{i});
        %counts how many times each letter in letterList occurs in w
    end

    H = 0; %set entropy to zero
    for i = 1:(counter - 1) %same as previous for loop
        P_i = letterTotal(i)/alphabetSize;
        %calculate the probability of occurance of each letter
        H = P_i*log(P_i) + H;
        %sum of the probabilities and logarithms for each letter
    end

    M = -H; %make the values positive
end
```

2.2 ComplexityFunction

Definie a function ComplexityFunction(w,n) that computes the complexity function on the sequence w for the subword length n.

This function consists of a single for loop. It outputs a number representing the number of subwords in w. Again, the prebuilt functions `extractBetween`, `extractBefore`, and `contains` are used. The number of possible subwords is determined by subtracting the length of the word w by the length of each subword, n, and 1. The subwords of length n are extracted from w using `extractBetween`. The functions `extractBefore` and `contains` are used to verify if each subword is unique. Note that this function is very similar to the MetricEntropy function.

```
function C = ComplexityFunction(w, n)
%note that w must be a character array

    totalSubwords = strlength(w) - (n - 1);
    %length of subwordList
    counter = 0; %count occurances of each subword
    for i = 1:totalSubwords
        subword = extractBetween(w,i,(n + i - 1));
        %n length subword betinning at i
        prevWords = extractBefore(w, (n + i - 1));
        %creates character array of all values of w before end of subword
        if contains(prevWords,subword) == 0
            %checks if prevWords contains subword
            counter = counter + 1; %increase counter
        end
    end

    C = counter; %counter holds one more value than number of subwords
end
```

2.3 TopologicalEntropy

Define a function TopologicalEntropy(w) that computes the topological entropy of w.

The length of the string w is determined by using the `strlength` function which "returns the number of characters in `str`"[4]. Next, the inequality is solved using a while loop. Essentially, $L = 0$ when one or both of the sides of the inequality are false and $L = 1$ when they are both true. The loop is exited when this occurs, yielding the value of n that satisfies the inequality. Note that we must manually calculate log base 4 by dividing the natural log of p and the natural log of 4.

```
function T = TopologicalEntropy(w)
%note that w must be a character array

    W = strlength(w);

    L = 0; %set logical counter to 0
    n = -1; %set n at -1 so we dont have to subtract 1 later
    while L == 0 %when the inequality is true exit the loop
        n = n + 1; %incrument n each itteration of the while

        n1 = 4^n + n - 1; %define the first equation
        n2 = 4^(n + 1) + (n + 1) - 1; %define the second equation

        L = n1 <= W & W < n2; %check when this is true
        %L = 1 when this is true, thus exiting the loop
    end

    p = ComplexityFunction(w, n); %find the number of n-length subwords
    logP = log(p) / log(4); %calculate log base 4 of p
    %log_4 is calcualted manually as there is no built in function
    T = logP/n;

end
```

2.4 Importing Data

Download the following file, http://www.math.oregonstate.edu/ koslickd/DNASequences.fasta and read it into Matlab using fastaread().

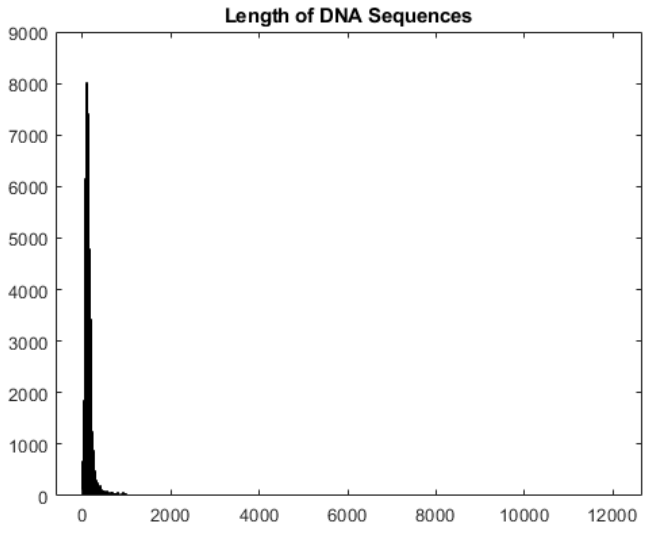
```
Data = fastaread('DNASequences.fasta'); %data input as structure array with 2
    dimensions
DNA = struct2cell(rmfield(Data,'Header'));
%rmfield extracts header field leaving only the dna sequences
%struct2cell converts structure array into cell array of characters
```

The code itself is fairly self explanatory. The comments provide additional information about the imported data.

Create a histogram of the lengths of the DNA sequences.

To construct a histogram of the lengths of the DNA sequences, we must find the length of each DNA sequence. Do do this, we fill an array with the corresponding length of each sequence using the `strlength` function.

```
length = [];
for i = 1:58286
    length(i) = strlength(DNA{i}); %find length of each character array
end

histogram(length)
title('Length of DNA Sequences')
```



Note that the majority of the sequences are of length between 0 and 2000 with a large spike around 50. However, the histogram extends out towards 12000, indicating there is at least one sequence almost that long.

Calculate the metric entropy for each sequence in the file. Create a histogram of the metric entropy values.

```
mEntropy = [];
for i = 1:58286
    mEntropy(i) = MetricEntropy(DNA{i},length(i));
        %calculate metric entropy for each DNA sequence
        %length corresponds to each DNA sequence
end

histogram(mEntropy)
title('Metric Entropy')
```

Again the code is fairly self explanatory with the comments. The histogram is located in the conclusions section next to the histogram of topological entropies.

Select the sequences that have metric entropy less than 1.2 (that is, the bottom 1.43% of the entropies) and save them to a file.

The MetricEntropy function described earlier was used to calculate the metric entropy of each DNA sequence. The histogram is located in the conclusions section next to the histogram of topological entropies.

```
subsetMEntropy = mEntropy < 1.2; %create a logical array where this is true
%1 means the inequality is true, 0 means it is false
%find seqnences with metric entropy less than 1.2 (bottom 1.43%)
subsetDNA1 = {}; %empty character array to hold such sequences
counter = 1;
for i = 1:58286
    if subsetMEntropy(i) == 1 %whenever this is true
        subsetDNA1{counter} = DNA{i}; %take that value and put it into the new
            vector
        counter = counter + 1;
    end
end

save ('subsetDNA1.mat', 'subsetDNA1') %save to a .mat file
```

To find the DNA sequences with metric entropy less than 1.2, we first create a logical array and input all values that satisfy the array into a new array. This array was then exported into a .mat file which consisted of 1x836 cells.

Calculate the topological entropy for each sequence in the file. Create a histogram of the topological entropy values.

```
tEntropy = [];
for i = 1:58286
    tEntropy(i) = TopologicalEntropy(DNA{i});
    %calculate topological entropy for each DNA sequence
end

histogram(tEntropy)
title('Topological Entropy')
```

The topological entropy was computed in the same way the metric entropy was computed using the TopologicalEntropy function described earlier. Again, the histogram is located in the conclusion section next to the histogram of metric entropies.

Select the sequences that have topological entropy less than .75 (that is, the bottom 1.44% of the entropies) and save them to a file.

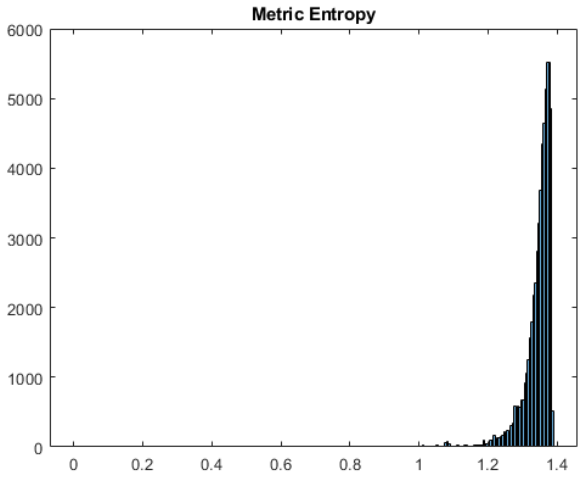
```
subsetTEntropy = tEntropy < 0.75;
%create a logical array where this is true
%1 means the inequality is true, 0 means it is false
%find sequences with topological entropy less than 0.75 (bottom 1.44%)
subsetDNA2 = {}; %empty character array to hold such sequences
counter = 1;
for i = 1:58286
    if subsetTEntropy(i) == 1 %whenever this is true
        subsetDNA2{counter} = DNA{i}; %take that value and put it into the new
            vector
        counter = counter + 1;
    end
end

save('subsetDNA2.mat', 'subsetDNA2') %save to a .mat file
```

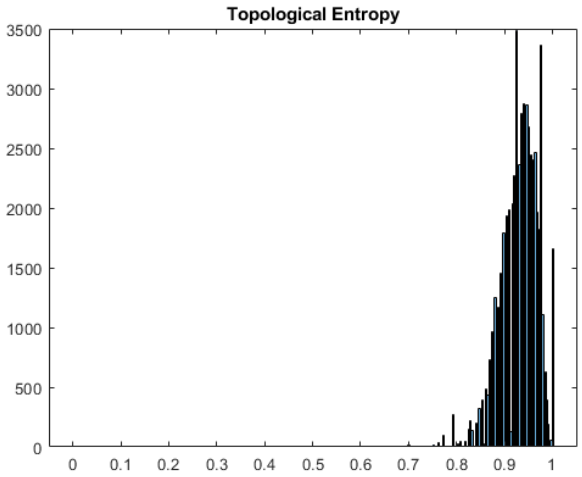
To find the DNA sequences with topological entropy less than .75, we follow the same procedure as for metric entropies. Again we save to a .mat file which contains a 1x33 cell of DNA sequences.

3 Conclusion

Compare and contrast the histograms of metric entropies and topological entropies.

Below we can clearly see the differences in the two histograms. Recall metric entropy is not necessarily bounded by 0 and 1. In fact, almost all of the values are between 1 and 1.4. Topological entropy is bounded, though most of the values again fall closer to 1. The histogram of the metric entropies is smooth whereas the histogram of topological entropies has many spikes. Both follow an overall trend of slowly increasing to a point and then sharply decreasing after reaching the peak. Lastly, note that there is a higher concentration of sequences with similar metric entropies. This is indicated by the y-axis which extends up to 6000. The y-axis for topological entropy only extends to 3500.





Consider the low metric entropy and low topological entropy sequences that you saved; what observations can you make about what it means for a sequence to have low metric/topological entropy?

There are 836 entries in the low metric entropies file and 33 entries in the low topological entropies file. This is interesting because these lists consist of the bottom 1.43% of metric entropies and the bottom 1.44% of topological entropies. This indicates that it is much rarer for a sequence to have low topological entropy than it is to have a low metric entropy. Although this may seem contradictory (we begin with the same number of DNA sequences) it is not. Recall we are calculating the bottom percent of topological and metric entropies, not lengths. Therefore, we may have many different sequences with the same topological entropies. So, it is not unusual that the sizes of each file are different. Furthermore, the sequences in the low metric entropy file tend to be significantly longer than the sequences in the topological entropy file. This may be because we can compare topological entropies of sequences of different lengths. This oculd leads to a larger variation in sequences with low topological entropy.

References

[1] Koslicki, D. *Topological entropy of DNA sequences* Bioinformatics, **27** No.8, (2011), pp. 1061–1067

[2] Heath, Ralph C. *Topological entropy* Trans. of the Amer. Math. Soc., **114** No.2, (1965), pp. 209–319

[3] Shannon, C.E. *A mathematical theory of communication* Bell Sys. Tech. J., **27** No.3, (1948), pp. 379–423

[4] Matlab 2019b, The MathWorks, Inc., Natick, Massachusetts, United States.